NASA Technical Memorandum 109054

A DATAFLOW ANALYSIS TOOL FOR PARALLEL PROCESSING OF ALGORITHMS

Robert L. Jones III

November 1993

NASA
National Aeronautics and Space Administration

Langley Research Center
Hampton, Virginia 23665

A DATAFLOW ANALYSIS TOOL FOR PARALLEL PROCESSING OF ALGORITHMS

Robert L. Jones III
NASA Langley Research Center
Hampton, Virginia

SUMMARY

A graph-theoretic design process and software tool is presented for selecting a multiprocessing scheduling solution for a class of computational problems. The problems of interest are those that can be described using a dataflow graph and are intended to be executed repetitively on a set of identical parallel processors. Typical applications include signal processing and control law problems. Graph analysis techniques are introduced and shown to effectively determine performance bounds, scheduling constraints, and resource requirements. The software tool is shown to facilitate the application of the design process to a given problem.

1. INTRODUCTION

This paper describes methods capable of determining and evaluating the steady-state behavior of a class of computational problems for iterative parallel execution on multiple processors. The computational problems must be capable of being described by a directed graph. When the directed graph is a result of inherent data dependencies within the problem, the directed graph is often referred to as a dataflow graph. Dataflow graphs, being generalized models of computation, have been getting increased attention for use in modeling parallelism inherent in computional problems [1-3].

Within the context of this paper, graph nodes represent schedulable tasks and graph edges represent the data dependencies between the tasks. Since the data dependencies imply a precedence relationship, the tasks make up a partial-order set. That is, some tasks must execute in a particular order whereas other tasks may execute independent of other tasks. When a computational problem or algorithm can be described using a dataflow graph, the inherent parallelism present in the algorithm can be readily observed and exploited. The modeling methods presented in this paper are applicable to a class of dataflow graphs where the time to execute tasks are assumed constant from iteration to iteration when executed on a set of identical processors. Also, it is assumed that the dataflow graph is data independent. That is, any

decisions present within the computational problem are contained within the graph nodes rather than described at the graph level. The dataflow graph provides both a graphical and mathematical model capable of determining run-time behavior and resource requirements at compile-time. In particular, it will be shown how the dataflow graph analysis can determine the exploitable parallelism, theoretical performance bounds, speedup, and resource requirements of the system. Since the graph edges imply data storage, the resource requirement specifies the minimum amount of memory needed for data buffers as well as the processor requirements. Obtaining this information is useful in allowing a user to match the resource requirements with resource availability. In addition, the dynamic, nonpreemptive scheduling and synchronization of the tasks that is sufficient to obtain the theoretic performance is specified by the dataflow graph. This property allows the user to direct the run-time execution according to the dataflow firing rules (*i.e.*, when tasks are enabled for execution) so that the run-time effort is reduced to simply allocating an idle processor to an enabled task [4-5]. When resource availability is not sufficient to achieve optimum performance, tasks can be moved within a range dictated by the dataflow analysis [6]. Predicting the computing performance, resource requirements, and processor utilization connected with the execution of a dataflow graph requires the determination of steady-state behavior. Dataflow graph analysis concepts discussed in this paper are capable of determining the earliest execution times and laxity, or slack time, for all tasks under steady-state conditions.

As for any mathematical model, there is a need for efficient software tools which facilitate the use of the model in solving problems. A software tool developed to implement the design and analysis methods described in this paper is presented. The software program is referred to as the Design Tool and is shown to provide automatic and user-interactive analysis capabilities applicable to the design of a multi-processing solution. The motivation behind this Design Tool was driven by research toward the adaptation of multiprocessing computations to emerging new space-qualified hardware for aerospace applications. The research has resulted in the development of a multiprocessing operating system based on a directed-graph approach called the ATAMM Multicomputer Operating System (AMOS) which is based on the ATAMM (Algorithm to Architecture Mapping Model) [7]. The Design Tool was developed not only to interface with the ATAMM application-development environment presented in [5]

and [7] but to serve the needs of other potential dataflow applications as well. For example, the design procedures based on ATAMM have been shown to solve signal processing problems addressed by Parhi and Messerschmitt [3], [8]. Also, information provided by the Design Tool could be used as scheduling constraints as done in [6] to drive other scheduling algorithms. Even though it is not discussed in this paper, the Design Tool has the capability to impose artificial data dependencies between tasks. Such artificial data dependencies have been shown to be a viable technique for improving performance or making performance tradeoffs to match resource requirements, due to the exposed parallelism, with resource availability [9-10].

The modeling of a computational problem with a dataflow graph and analysis diagrams is discussed in Section 2. Also, forward- and backward-search techniques are discussed and shown to determine the scheduling range for tasks. Performance metrics and resource requirements procedures implemented in the Design Tool are also presented in Section 2. The Design Tool displays and features are presented in Section 3 and conclusions are summarized in Section 4. The use of brand names in this document is for completeness and does not imply NASA endorsement.

2. DATAFLOW GRAPHS AND PERFORMANCE METRICS

A computational problem (job) can often be decomposed into a set of tasks to be scheduled for execution [11]. If the set of tasks are not independent of one another, there will be a precedence relationship imposed on the tasks in order to obtain correct computational results. A task system can be represented formally as ($\mathcal{T}$, $\prec$, $\mathcal{L}$, $\mathcal{M}_0$) where

$\mathcal{T}$ = { T_1, T_2, T_3, ... T_n } is a set of n tasks to be executed,

$\prec$ is the precedence relationship on $\mathcal{T}$ such that $T_i \prec T_j$ signifies that T_j cannot execute until the completion of T_i,

$\mathcal{L}$ = { L_1, L_2, L_3, ... L_n } is a set of run-time latencies such that task T_i takes L_i amount of time to execute, and

$\mathcal{M}_0$ is the initial state of the system, as indicated by the presence of initial data.

Such task systems can be described by a directed graph where nodes (vertices) represent the tasks and edges (arcs) describe the precedence relationship between the tasks. When the precedence constraints given by $\prec$ are a result of the dataflow between the tasks, the directed graph is referred to as a dataflow graph (DFG) as shown in Figure 1. Special transitions called sources and sinks are also provided to model the input and output data streams of the task system. The presence of data is indicated within the DFG by the placement of tokens. The DFG is initially in the state indicated by the initial marking $\mathcal{M}_0$. The graph transitions through other markings as a result of a sequence of node firings. That is, when a token is available on every input edge of a node and sufficient resources are available for the execution of the task represented by the node, the node fires. When the node associated with task T_i fires, it encumbers one token from each of its input edges, delays an amount of time equal to L_i, and then deposits one token on each of its output edges. Sources and sinks have special firing rules in that sources are unconditionally enabled for firing and sinks consume tokens, but do not produce any. By analyzing the DFG in terms of its critical path, critical circuit, dataflow schedule, and the token bounds within the graph, the performance characteristics and resource requirements can be determined *a priori*. The Design Tool depends on this dataflow representation of a task system, and the graph-theoretic performance metrics presented in this paper.

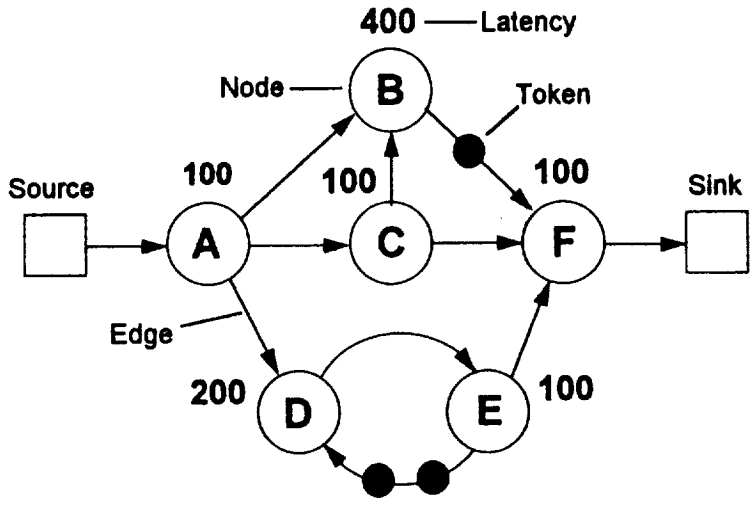

Figure 1. Example dataflow graph.

The two types of concurrency that can be exploited in dataflow algorithms can be classified as parallel and pipeline. Parallel concurrency is associated with the execution of tasks that are independent (no precedence relationship imposed by $\prec$), whereas pipeline concurrency is associated with the iterative execution of the algorithm for successive data packets without waiting for earlier data packet iterations to complete.

The extent to which parallel concurrency can be exploited is dependent on the number of parallel paths and the availability of resources required to exploit the parallelism. The critical path within the dataflow graph determines the TBIO (time between input and output) defined as the time between source input and the corresponding sink output. If there are no initial tokens present in the DFG, TBIO can be determined using the traditional critical path analysis where TBIO is given as the sum of latencies in $\mathcal{L}$ along the critical path. Under such conditions the minimum time in which all tasks can be executed for a single iteration, given sufficient resources, is defined as the schedule length, ω, and will be equal to TBIO. However, when $\mathcal{M}_0$ defines initial tokens in the forward direction, it will be shown in the next section that ω may be greater than TBIO. Cases such as this include many signal processing and control algorithms where initial tokens are expected to provide previous state information (history) or to provide data delays within the algorithm. For the example shown in Figure 2, the task output z(n) is dependent on input x(n), input y(n - α) from the previous αth iteration, and output z(n - β) from the previous βth iteration. Implementation of this function would require α initial tokens on the y(n - α) edge and β initial tokens on the z(n - β) edge in order to create the desired delays. In such cases, the critical path and thus TBIO is also dependent on the iteration period. The iteration period is specified as the time between successive sink outputs (TBO). For example, given that a node fires when all input tokens are available, assuming sufficient resources, the earliest time at which the node shown in Figure 2 could fire, referred to as the earliest start (ES) time, would be dependent on the longest path latency leading to either the x(n) or y(n - α) edge. Assuming that the α and β tokens are the only initial tokens within the graph, the time it would take a token associated with the nth iteration to reach the x(n) edge would equal the path latency leading to the x(n) edge. Likewise, the minimum time at which the "token" firing the nth iteration on the y(n - α) edge could arrive from the source equals the path latency leading to the y(n - α) edge. However, since

this "token" is associated with the (n - α)th iteration (produced α TBO intervals earlier), the actual path latency referenced to the same iteration is reduced by the product of α and TBO. From this example, it is easy to infer that the actual path latency along any path with a collection of N initial tokens is equal to the summation of the associated node latencies less the product of N and TBO. Thus, the critical path (and TBIO) is a function of the iteration period, TBO, and is given as the path from source to sink which maximizes the following equation for TBIO,

$$\text{TBIO} = \max\left[\left(\sum_{i \in L} L_i\right) - N * \text{TBO}\right] \quad \{\text{for all paths}\}. \tag{1}$$

$$z(n) = x(n) * y(n - \alpha) * z(n - \beta)$$

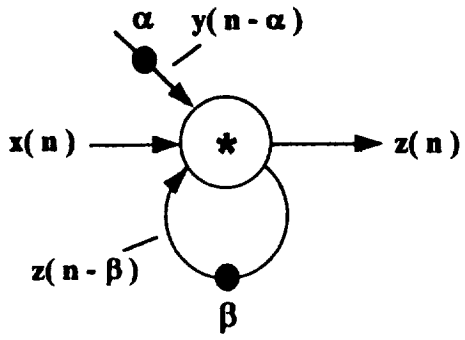

Figure 2. Example function implementation.

For example, application of Equation 1 to the dataflow graph of Figure 1 results in finding the critical path to be A ≺ D ≺ E ≺ F for a TBIO of 500 clock units and a TBO of 250 clock units. For the type of dataflow graphs targeted by the Design Tool, TBO will equal the input-injection period at steady state. Thus, predictable steady-state behavior can occur at these TBO and TBIO values when the input injection period is 250 clock units.

Assuming for the moment that no initial tokens are present, a latest finish time analysis would involve working backwards from all sinks and determining the latest time each task must complete in order to prevent an increase in the TBIO given by Equation 1. The latest finish (LF) time for a given task is equal

to the TBIO (for a given sink) less the minimum path latency to the task output from all paths leading backwards from the sink. The combination of ES and LF times provides the means to calculate the float or slack time that might be present for each task. Slack time indicates the maximum delay in task completion that can be tolerated without delaying the start times of successor tasks which result in an increase in TBIO. Slack time for a task is given by Equation 2 with latency L:

$$\text{slack time} = LF - ES - L. \tag{2}$$

In cases where edges have initial tokens, however, determination of ES and LF times requires additional consideration. Consider the graph fragment of Figure 3 which expresses an N-TBO delay relationship between tasks T_i and T_t.

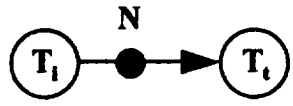

Figure 3. Dependent tasks with N-TBO delay.

For each such task dependency represented by Figure 3, the following time constraint is imposed:

$$LF(T_i) = ES(T_t) + TBO * N, \tag{3}$$

where $LF(T_i)$ represents the LF time of T_i due to the initial token(s), $ES(T_t)$ represents the ES time of T_t, and N is the number of initial tokens on the $T_i \prec T_t$ edge. Stated in words, Equation 3 determines the latest finish time of task T_i which returns a token on the edge initialized with N tokens such that the firing of task T_t will not be delayed.

Proof of Equation 3:

During the transient state, the graph will execute based on earliest start times, neglecting edges with initial tokens. However, since the next data packet will arrive one TBO interval later, an additional time constraint will be imposed if initial tokens exist in the graph. The node T_t with N

initial input tokens has the potential (depending on other input dependencies) of repeated firings until all N tokens are consumed. With each node firing with period TBO, the elapsed time to consume N tokens is the product of N and TBO. The predecessor node T_i must return a token within N * TBO time relative to the ES so that the next firing of T_t is not delayed. Therefore, in order for node T_i to generate its first token in this timely manner which maintains the task schedule defined by the first iteration, it must do so by the time determined by Equation 3. Otherwise, the firing of node T_t will be delayed □.

It has been shown that the minimum time in which tokens can propagate through a recurrence loop or circuit in one periodic cycle is given by Equation 4 [3], [6], [12-13],

$$T_o = \max \left\lceil \frac{C_i}{N_i} \right\rceil \quad \{\text{for all ith circuits}\}, \tag{4}$$

where C_i is equal to the summation of node latencies associated with the ith circuit and N_i is the number of initial tokens within the circuit. Thus, Equation 4 establishes a graph-imposed lower bound on TBO. Given a finite number of processors, R, the lower bound on the iteration period (or TBO_{lb}) is given by

$$TBO_{lb} = \max \left[T_o, \left\lceil \frac{TCE}{R} \right\rceil \right], \tag{5}$$

where TCE (total computing effort) is the sum of latencies in $\mathcal{L}$,

$$TCE = \sum_{i \in \mathcal{L}} L_i, \tag{6}$$

and T_o is zero if no recurrence loops are present in the DFG. Accordingly, the theoretically optimum value of R for a given TBO period, referred to as the calculated R, is given as

$$R = \left\lceil \frac{TCE}{TBO} \right\rceil. \tag{7}$$

Attaining a desired level of throughput (inverse of TBO) may require that some tasks with execution durations greater than TBO be multiply instantiated. A multiple instantiation of a task is defined as the

simultaneous execution of the same task on successive data packets by different processors. The number of instantiations required of each task T_i at run-time, for a given TBO, is bounded by Equation 8:

$$\text{Instantiations of } T_i = \left\lceil \frac{L_i}{TBO} \right\rceil. \tag{8}$$

With or without multiple instantiations of tasks, it is shown in [14] that every task executes once within a TBO interval with sufficient resources. Consequently, using Amdahl's Law, speedup (S) can be defined as

$$S = \frac{TCE}{TBO}, \tag{9}$$

and processor utilization, U, ranging from 0 to 1 can be determined by Equation 10:

$$U = \frac{S}{R}. \tag{10}$$

3. DATAFLOW DESIGN TOOL

A software tool is presented in this section which implements the graph analysis concepts discussed in the previous section. The software, referred to as the Dataflow Design Tool, was written in C++ by Borland International, Inc. for Microsoft Windows 3.1 by Microsoft Corporation. The software can be hosted on an i386/486 personal computer or compatible. The Design Tool takes input from a text file which specifies the topology and attributes of the DFG. A graph-entry tool has been developed to create the DFG text file. The various displays and features are shown to provide an automated and user-interactive design process which facilitates the selection of a multiprocessor solution based on dataflow analysis.

After loading a DFG description, the Design Tool will search the DFG for circuits in order to determine the minimum iteration period (T_0) using Equation 4. TBO will initially be set to the lower bound given in Equation 5. The calculated R will initially be given by Equation 7. Graph analysis algorithms are implemented by the tool to provide an analytical determination of steady-state TBIO,

dataflow-schedule range, ω, and run-time resource requirements. Any changes to TBO, R, or $\prec$ results in a re-application of the analytical-analysis procedures, generating a new solution.

The dataflow graph shown in Figure 1 will be used in this section for the purposes of presenting the key Design Tool performance displays and demonstrating the dependence that steady-state behavior has on $\prec$, $\mathcal{M}_O$, and TBO.

The Design Tool has a user-interface panel, referred to as the Metrics window as shown in Figure 4, containing buttons and menus for the purpose of displaying performance bounds, setting TBO and R, or invoking the various graphic displays. The time measurements shown in the Design Tool windows are given in *clock units* so that the resolution of the measurement can be user-interpreted.

Upon analyzing the DFG, the Design Tool has determined that TCE is 1000 clock units. Due to the critical path A $\prec$ D $\prec$ E $\prec$ F, the lower bound on TBIO ($TBIO_{lb}$) has been determined to be 500 clock units even though at the current TBO of 250 clock units, ω (shown next to the *Schedule* button) is 600 clock units. The TBO_{lb} has been calculated to be 250 clock units based on the critical circuit consisting of nodes D and E. The calculated R is determined to be 4 which is the optimum number of processors for repetitive, steady-state execution at the given TBO and TBIO.

Also shown in Figure 4 is a Gantt chart display of steady-state execution for a single data packet assuming unlimited resources so that all of the inherent parallelism of the decomposed algorithm is exposed. The Gantt chart is referred to hereafter as a Single Graph Play (SGP) diagram. Shaded bars are used to indicate the earliest start and execution duration of each task according to the dataflow analysis. As expected, Figure 4 shows that task B is schedulable only after the completion of tasks A and C. Slack time is shown as unshaded bars where task B and C are shown to have finite slack. The slack time of task B is wrapped-around to both signify that the slack is due to inter-iteration dependence with task F and avoid expanding the time-line beyond ω.

Individually controlled left and right cursors (solid vertical lines) are provided for taking time measurements. The left and right cursors are shown in Figure 4 at the far left and at the completion of task F, respectively, for the purpose of measuring TBIO. The left cursors, measuring relative time, indicates the start of task A to be at time zero relative to the injection of a given data packet (the "0"

next to "TIME" at the bottom of the display). The differential time between the left and right cursors is displayed in parentheses. The elapsed time is measured to be 500 clock units, agreeing with the $TBIO_{lb}$ shown in the Metrics window.

Pressing the *Performance* button invokes a window shown in Figure 5 displaying the graph-theoretic speedup potential. The display expands or shrinks the abscissa each time the number of processors (R) is increased or decreased, respectively. Figure 5 indicates that maximum speedup performance is graph limited at 7 processors; additional processors will not result in any further speedup. As pointed out in the previous section, this leveling-off of performance is attributable to the recurrence loop (circuit) within the DFG. Without this circuit, the graph-theoretic speedup would continue to increase linearly with the addition of processors. Realistically speaking, however, this linear increase in speedup would ultimately be adversely affected by operating system overhead, such as synchronization costs and inter-processor communication.

The periodic graph execution for multiple data packets can be portrayed in another Gantt chart referred to as a Total Graph Play (TGP) diagram. The TGP window shown in Figure 6 portrays, as a snapshot in time, the execution over a single periodic interval of length TBO. Like the SGP, the TGP represents task executions using bars. Overlapped bars for a given task indicates that the task has multiple instantiations as is the case for task B in Figure 6. Constructing the steady-state TGP diagram is accomplished by mapping the ES times (relative to the SGP diagram) to a time interval of width TBO using the mapping function ES *mod* TBO where *mod* TBO returns the remainder after dividing by TBO. The iteration period is shown in parentheses to be equal to the current TBO of 250 clock units. The TGP-view of graph execution portrays not only the parallel concurrency that is being exploited but also the pipeline concurrency that results from the simultaneous execution of different data packets within the graph. The degree of pipeline concurrency is indicated by the maximum number of data packets (referred to as $\mathcal{P}$) in the graph in any TBO interval. This metric is given by the smallest integer greater than the ratio of the schedule length, ω, to TBO as shown in Equation 11:

$$\mathcal{P} = \left\lceil \frac{\omega}{\text{TBO}} \right\rceil. \tag{11}$$

The TGP bars are numbered and shaded according to relative data packet numbers 1 to $\mathcal{P}$ such that a data packet numbered $\mathcal{D}$ will refer to a data packet injected into the graph one TBO interval after a data packet numbered $\mathcal{D}$-1. The inter-iteration dependency between tasks B and F is apparent by noticing in Figure 6 that shortly after task B completes execution of a given data packet (data packet 1), task F begins execution of a data packet (data packet 2) injected into the graph one-TBO interval later. As discussed previously, this inter-iteration dependency results from the one-TBO delay caused by the initial data token. By counting the maximum overlap of TGP bars in Figure 6, it is evident that 4 processors are required with 100% utilization for dynamic scheduling at this TBO periodicity.

A summary of the task system ($\mathcal{T}$, $\prec$, $\mathcal{L}$, $\mathcal{M}_0$) is given by a window referred to as the Graph Summary window shown in Figure 7 requiring 4 processors for a TBO equal to 250 clock units. The Graph Summary window displays the $\mathcal{L}$, ES times, LF times, slack, and instantiations (INST) for each task in $\mathcal{T}$ as portrayed by the SGP and TGP windows. Since the dataflow edges imply physical storage of data shared among tasks, the Graph Summary window also indicates minimum memory requirements (neglecting fault tolerant issues) for each data edge in $\prec$. The calculation of memory in terms of edge buffers is based on a condition which bounds the number of tokens which accumulate on a DFG edge during execution. The condition assumes that each data edge would be paired with an acknowledgment edge, signifying when an immediate successor node (task) has encumbered an output token (data) for use as input. Such an acknowledgment edge would allow the condition which enables a dataflow task for execution (arrival of input data) to include the availability of an empty data buffer so that memory represented by each data edge can be allocated statically, eliminating the need for dynamic allocation of memory at run-time while still assuring that data will not be overwritten. Thus, in addition to the output full (OF) buffers that are required for initial data, a finite number of output empty (OE) buffers may be needed to achieve the scheduling solution given by the Design Tool. The summation of OE and OF, given as QUEUE in Figure 7, specifies the total number of output-edge buffers required for synchronized sharing of data between dependent tasks. For example, the edge from node B to F requires buffering of 2 data packets.

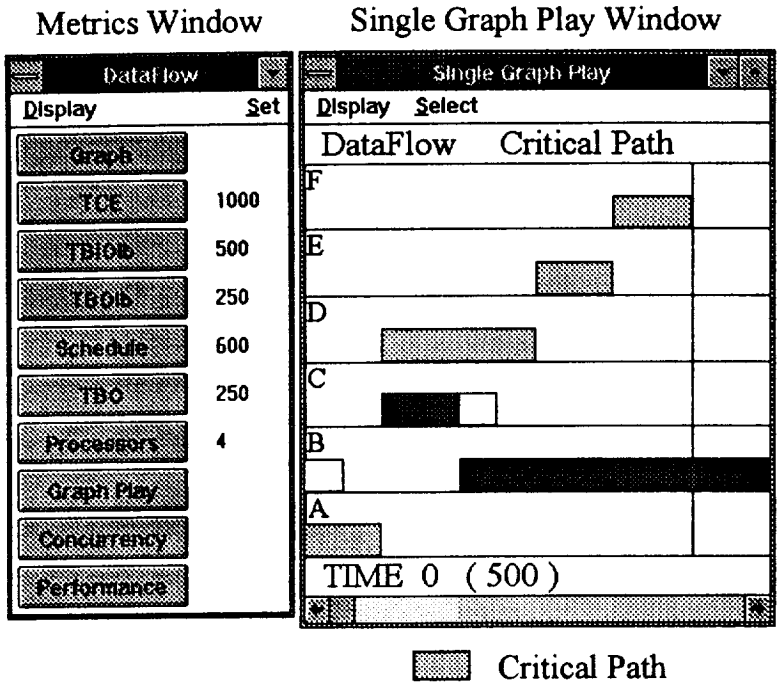

Figure 4. Dataflow schedule of the DFG in Figure 1 for a speedup of 4.

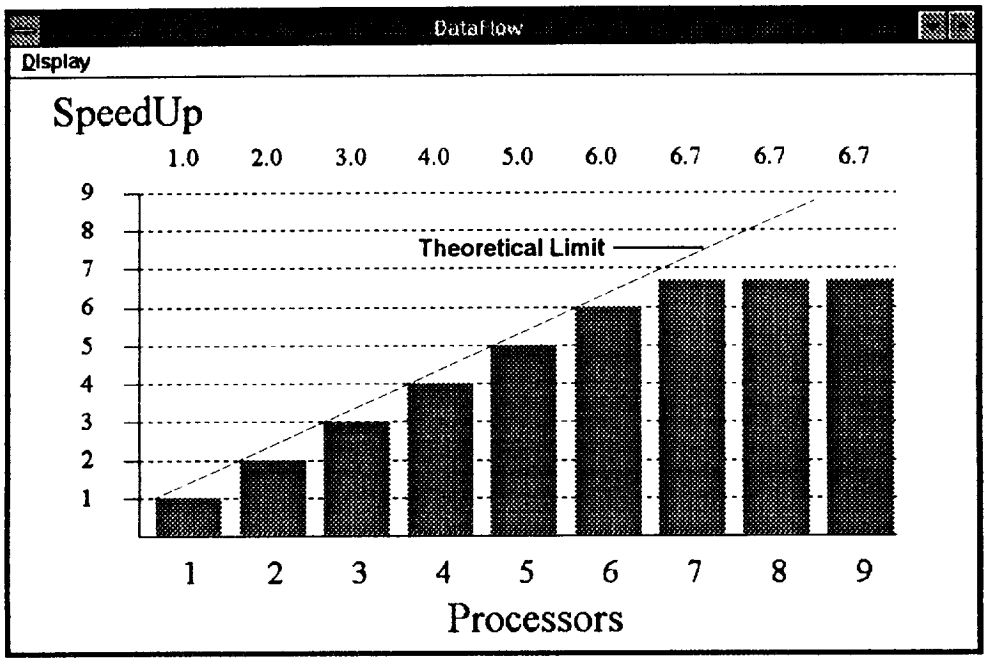

Figure 5. Speedup potential of the DFG in Figure 1.

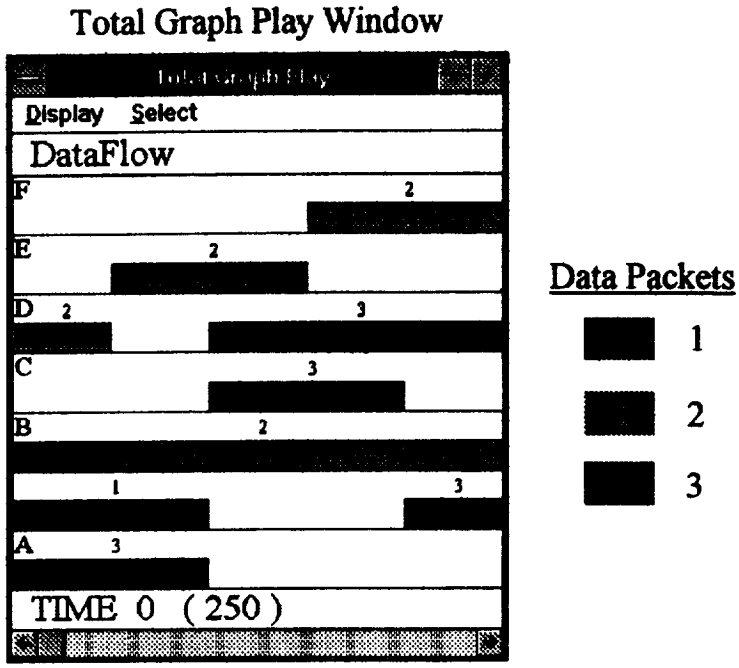

Figure 6. Periodic schedule of the DFG in Figure 1 for a speedup of 4.

Dataflow Summary

Display

NAME	LATENCY	ES	LF	SLACK	INST	OE/OF	QUEUE
A	100	0	100	0	1	1/0 -> D	1 -> D
						1/0 -> C	1 -> C
						1/0 -> B	1 -> B
B	400	200	650	50	2	1/1 -> F	2 -> F
C	100	100	250	50	1	1/0 -> B	1 -> B
						2/0 -> F	2 -> F
D	200	100	300	0	1	1/0 -> E	1 -> E
E	100	300	400	0	1	1/0 -> F	1 -> F
						0/2 -> D	2 -> D
F	100	400	500	0	1	1/0 -> Snk	1 -> Snk

Figure 7. Graph summary of Figure 1 for a speedup of 4.

The dependence that the critical path and steady-state behavior has on TBO when initial tokens are present within the DFG, as modelled in Equation 1, was discussed in the previous section. To demonstrate this behavior with an example, Figure 8 shows the performance characteristics and SGP window for the smallest possible TBO of 150 clock units, requiring 7 processors. This results in ω equal to 600 clock units which is still greater than the graph's TBIO; however, the critical path has changed from the previous example, now found to be $A \prec C \prec B \prec F$. Also, the initial token at this TBO performance has caused task F to delay 50 clock units after the completion of task E (measured by the SGP window cursors and displayed in parentheses), resulting in a TBIO equal to 550 clock units.

Shown alongside the TGP window of Figure 9, is a graphical portrayal of processor utilization that the Design Tool provides called a Total Resource Envelope diagram. The diagram plots the number of processors utilized at any given time within a periodic TBO interval. The shaded area under the processor curve is equal to the TCE of Equation 6. Accompanying the Total Resource Envelope window is a Utilization window which not only displays total utilization (shown to be 95.2%) but also the percentage of time a given number of processors is utilized. In this example, the Utilization window shows that up to 6 processors are utilized constantly whereas a 7th is utilized only 66.7% of the time.

Figure 10 shows the Graph Summary window required to achieve the dataflow scheduling and performance of Figure 8 and Figure 9. The LF of task F with no slack indicates that the TBIO is 550 clock units. Also, tasks B and D require 3 and 2 instantiations, respectively. As one might have expected, the queue sizes (memory requirements) have increased from the lower speedup example with 4 processors.

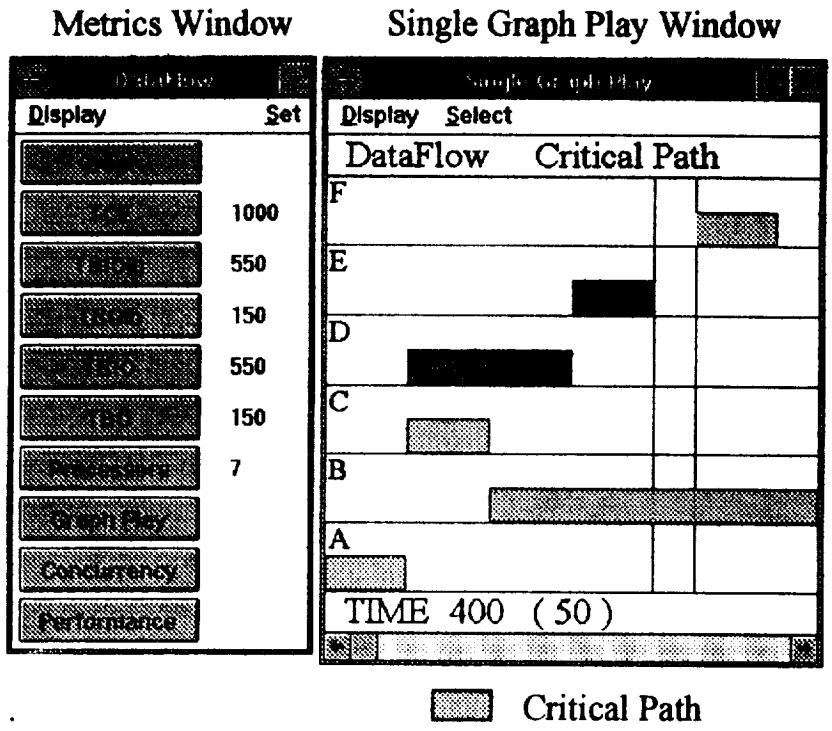

Figure 8. Dataflow schedule of the DFG in Figure 1 for a speedup of 6.7.

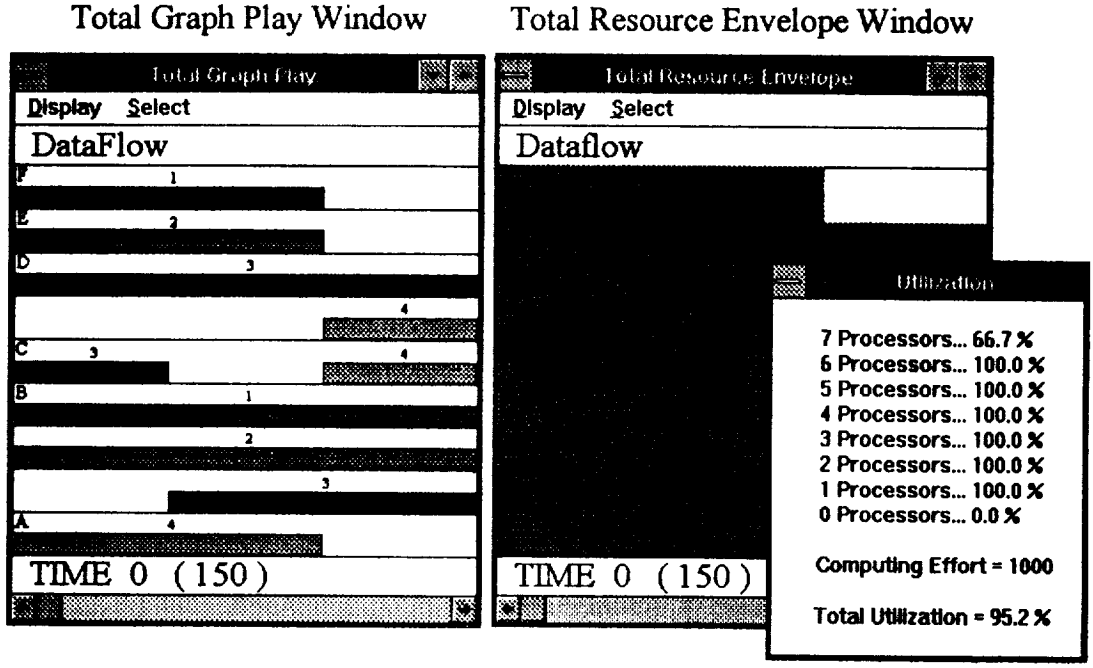

Figure 9. Periodic schedule of the DFG in Figure 1 requiring 7 processors.

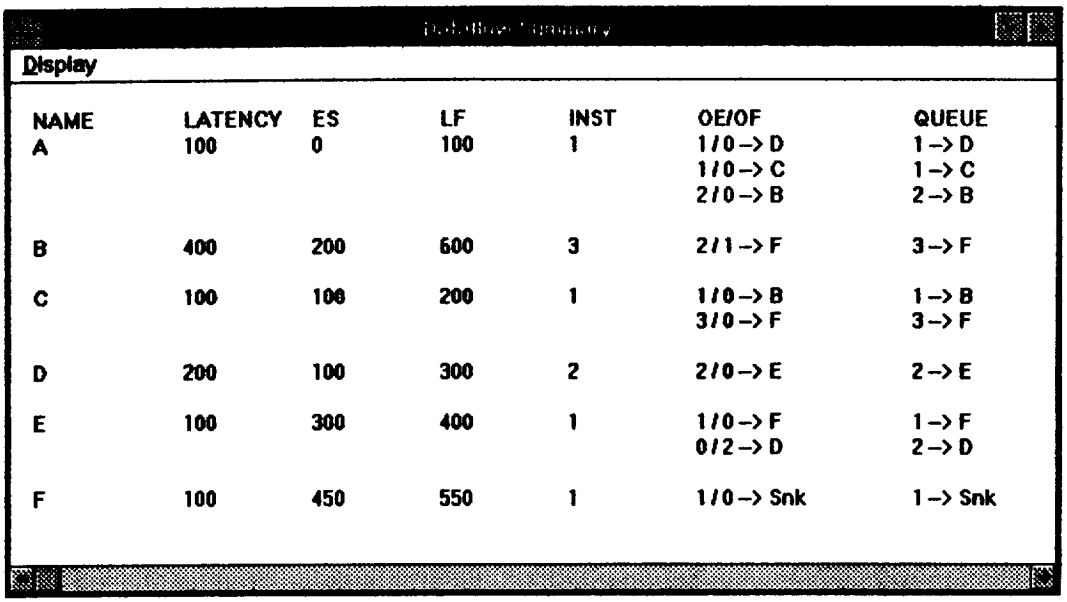

Figure 10. Graph summary of Figure 1 for a speedup of 6.7.

4. CONCLUSIONS

Dataflow graph analysis concepts were presented in this paper and shown to determine performance bounds inherent in a decomposed algorithm. A software program called the Design Tool was presented which implements an analytical dataflow analysis for determining graph-theoretic performance bounds, providing Gantt chart portrayals of scheduling behavior, and plotting processor utilization. The automated and user-interactive features of the tool were shown to facilitate the selection of a multiprocessing solution for dynamic scheduling. Scheduling ranges, earliest start and laxity of tasks, based on dataflow analysis can also be used as correctness criteria for static-scheduling algorithms.

ACKNOWLEDGMENT

The Design Tool has benefited from numerous discussions with Sukhamoy Som, Rodrigo Obando, Paul Hayes, and Asa Andrews. The Graph-Entry Tool was developed by Asa Andrews of CTA INCORPORATED, Hampton, Virginia.

REFERENCES

[1] Akshay K. Deshpande and Krishna M. Kavi, "A Review of Specification and Verification Methods for Parallel Programs, including the Dataflow Approach," *Proceedings of the IEEE*, vol. 77, December 1989, pp. 1816-1828.

[2] David E. Culler and Arvind, "Resource Requirements of Dataflow Programs," 15th Annual International Symposium on Computer Architectures, May 1988.

[3] Parhi, Keshab K.; and Messerschmitt, David G.: Static Rate-Optimal Scheduling of Iterative Data-Flow Programs via Optimum Unfolding. *IEEE Transactions on Computers*, vol. 40, Feb. 1991, pp. 178-195.

[4] P. J. Hayes, R. L. Jones, H. F. Benz, A. M. Andrews, and M. R. Malekpour, "Enhanced ATAMM Implementation on a GVSC Multiprocessor," *GOMAC '92 / 1992 Digest of Papers*, Nov. 9-12, 1992.

[5] Robert L. Jones, John W. Stoughton, and Roland R. Mielke, "Analysis Tool for Concurrent Processing Computer Systems," *IEEE Proceedings of the Southeastcon '91*, vol. 2, April 8-10, 1991, pp. 620-625.

[6] Sonia M. Heemstra de Groot, Sabih H. Gerez, and Otto E. Herrmann, "Range-Chart-Guided Iterative Data-Flow Graph Scheduling," *IEEE Transactions on Circuits and Systems*, vol. 39, May 1992, pp. 351-364.

[7] R. Mielke, J. Stoughton, S. Som, R. Obando, M. Malekpour, and B. Mandala, "Algorithm to Architecture Mapping Model (ATAMM) Multicomputer Operating System Functional Specification," NASA CR 4339, Nov. 1990.

[8] Matthew Storch, "A Comparison of Multiprocessor Scheduling Methods for Iterative Data Flow Architectures," NASA CR 189730, Feb. 1993.

[9] S. Som, R. R. Mielke, and J. W. Stoughton, "Effects of Resource Saturation in Real-Time Computing on Data Flow Architectures," *Twenty-Fifth Asilomar Conference on Signals, Systems and Computers*, vol. 1, Nov. 4-6, 1991, pp. 39-43.

[10] S. Som, R. Obando, R. R. Mielke, and J. W. Stoughton, "ATAMM: A Computational Model for Real-Time Data Flow Architectures," *International Journal of Mini and Microcomputers*, vol. 15, 1993, pp. 11-22.

[11] E. G. Coffman, *Computer and Job-Shop Scheduling Theory*, John Wiley and Sons, Inc., 1976.

[12] S. Som, J. W. Stoughton, and R. R. Mielke, "Strategies for Concurrent Processing of Complex Algorithms," NASA CR 187450, Oct. 1990.

[13] Roland R. Mielke, John W. Stoughton, and Sukhamoy Som, "Modeling and Optimum Time Performance for Concurrent Processing," NASA CR 4167, Aug. 1988.

[14] R. L. Jones, P. J. Hayes, A. M. Andrews, S. Som, J. W. Stoughton, and R. R. Mielke, "Enhanced ATAMM for Increased Throughput Performance of Multicomputer Data Flow Architectures", *Proceedings of the NAECON '91*, vol. 1, May 20-24, 1991, pp. 238-244.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (*Leave blank*)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	November 1993	Technical Memorandum

4. TITLE AND SUBTITLE
A Dataflow Analysis Tool for Parallel Processing of Algorithms

5. FUNDING NUMBERS
233-01-03

6. AUTHOR(S)
Robert L. Jones III

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
NASA Langley Research Center
Hampton, Virginia 23681-0001

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
National Aeronautics and Space Administration
Washington, DC 20546-0001

10. SPONSORING / MONITORING AGENCY REPORT NUMBER
NASA TM-109054

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Unclassified - Unlimited

Subject Category 61

12b. DISTRIBUTION CODE

13. ABSTRACT (*Maximum 200 words*)

A graph-theoretic design process and software tool is presented for selecting a multiprocessing scheduling solution for a class of computational problems. The problems of interest are those that can be described using a dataflow graph and are intended to be executed repetitively on a set of identical parallel processors. Typical applications include signal processing and control law problems. Graph analysis techniques are introduced and shown to effectively determine performance bounds, scheduling constraints, and resource requirements. The software tool is shown to facilitate the application of the design process to a given problem.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Multiprocessing; Dataflow; Analytical tool; Signal processing	20
	16. PRICE CODE: A03

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	